Stefan Lange | empira Software GmbH

# WPF und Silverlight Architektur

## Tipps zur Struktur von Anwendungen

Stefan.Lange@empira.de

25.02.2010

# Agenda

- Schichtenmodell
- Silverlight und WPF Code Sharing
- Typische Architektur Fehler
- Model - View - ViewModel
- Synchronisation

# Die richtige Architektur

- Gibt es leider nicht!

- Es gibt immer mehrere gute Möglichkeiten
- Immer wieder ähnliche Fragestellungen
- Manche Ansätze haben sich gegenüber anderen im Laufe der Zeit mehr bewährt

# Diskussionsgrundlage

- Fiktive LOB-Anwendung als Diskussionsgrundlage
- Prinzipien erklären
- Maximale Wiederverwendbarkeit zwischen WPF und Silverlight
- Ohne den Einsatz konkreter Frameworks

BASTA! .NET, VISUAL STUDIO & MORE

# Rahmenbedingungen

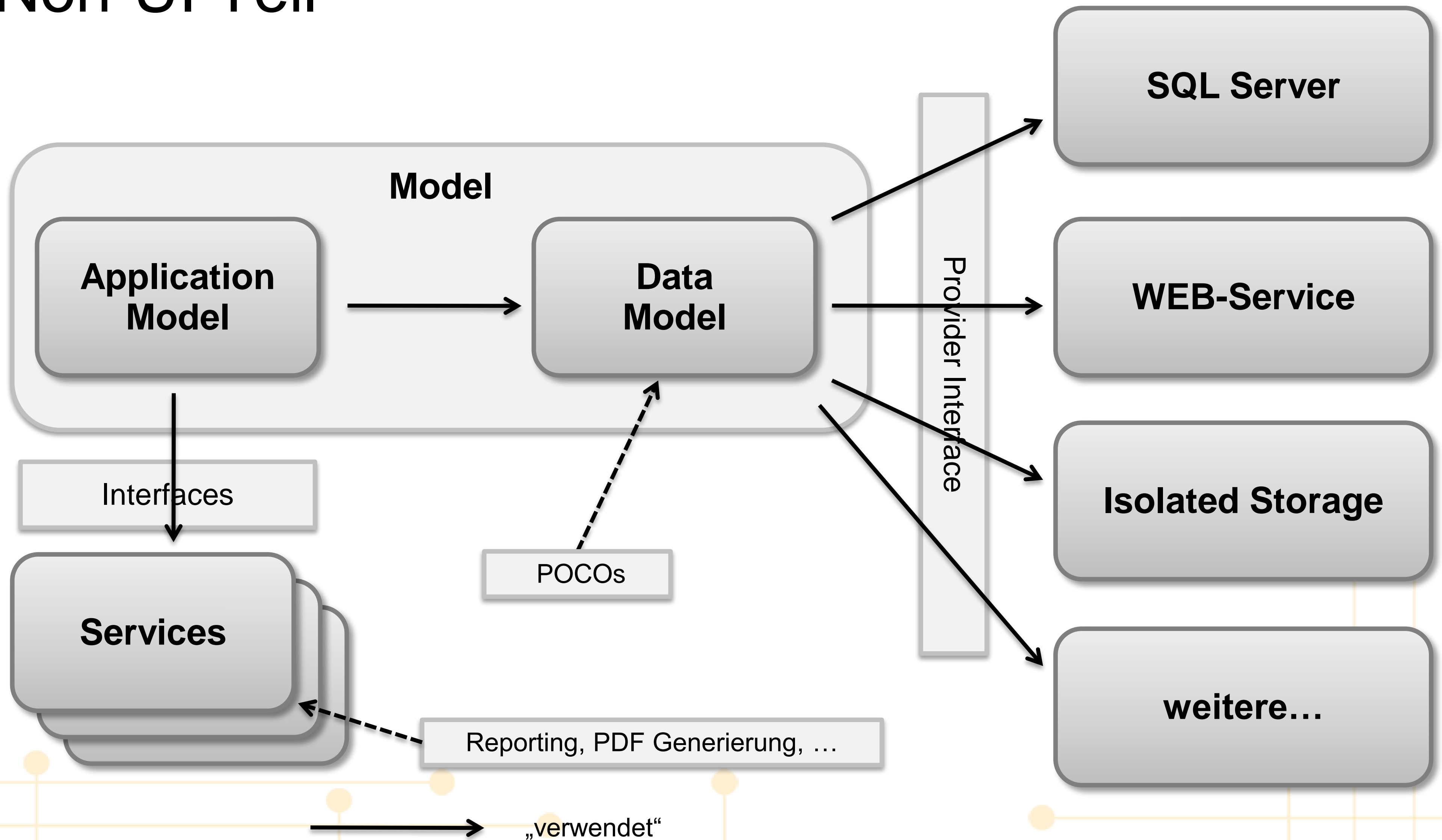
- LOB-Anwendung
- Modelliert Business-Cases
- Datenbank-basiert
- Kein ganz kleines Projekt als Ziel
- Keine „Spezialanwendung“
- Beschränkt auf WPF (Desktop) und Silverlight (Browser und Out-of-Browser)

# Aufteilung des Systems

- Userinterface
- UI-Datenanbindung
- ViewModel
- Kernapplikation
- Services
- Datenbank Layer

# Applikationslogik (Übersicht)

## Non-UI Teil



Data Model

# Data Model

- Auscodierte Klassen (POCOs)
- Einfache Datentransfer-Objekte, IDs
- I.d.R keine Vererbung
- Serialisierbar („WCF tauglich“)
- Nur innerhalb des Application Models verwendet
- Entkopplung von konkreter Implementierung
- In eigene Assembly legen

# Data Model Klasse (C#)

```
public class Employee : BaseClass
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public string TitleOfCourtesy { get; set; }

    …
}
```

BASTA! .NET, VISUAL STUDIO & MORE

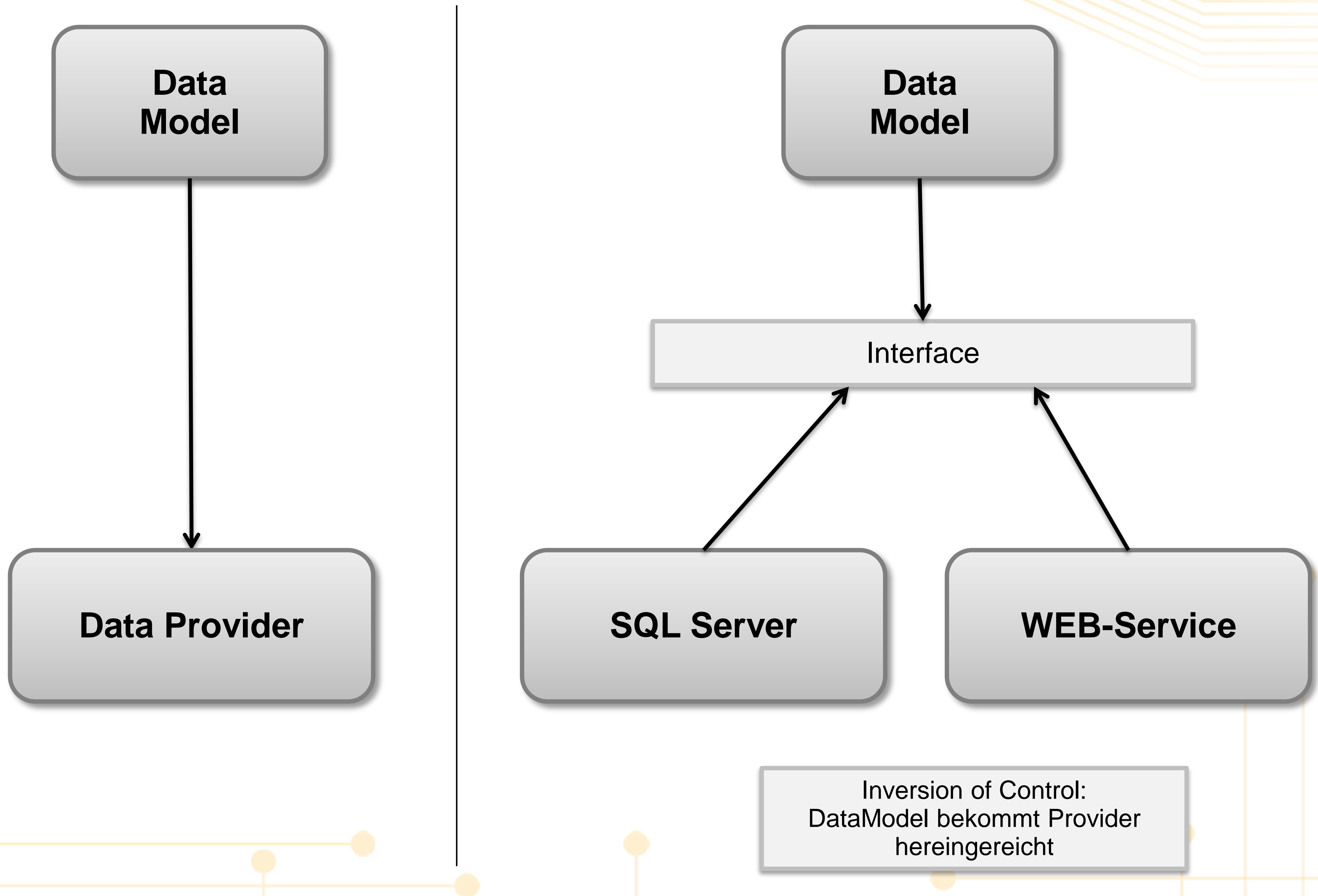
# Data Model Klasse (VB)

```
Public Class Employee
    Inherits BaseClass

Public Property FirstName() As String
    Get
      Return _FirstName
    End Get
    Set(ByVal value As String)
      _FirstName = value
    End Set
  End Property
  Private _FirstName As String

  Public Property LastName() As String
  …
End Class
```

- *auto-implemented* properties ab VB 2010

BASTA! .NET, VISUAL STUDIO & MORE

# Tipp: IOC verwenden!




BASTA! .NET, VISUAL STUDIO & MORE

# Frameworks

- Datenbankzugriff
  - NHibernate
  - ADO.NET Entity Framework (Version > 1!)
- Inversion of Control Container
  - MEF
  - Unity
  - Spring.NET
  - Castle Windsor

# BEISPIEL

- DataModel
- IDataProvider
  - AccessDataProvider
  - WcfServiceDataProvider

Application Model

# Application Model (I)

- Modelliert die Anwendungs-Domäne
- „Komplexere“ Klassen (i.d.R. nicht serialisierbar)
- Vererbung, zyklische Abhängigkeiten
- Schnittstelle nach außen
- Referenzen statt IDs
- In eigene Assembly legen
- Ggf. mehrere Models (nachladbar)

BASTA! .NET, VISUAL STUDIO & MORE

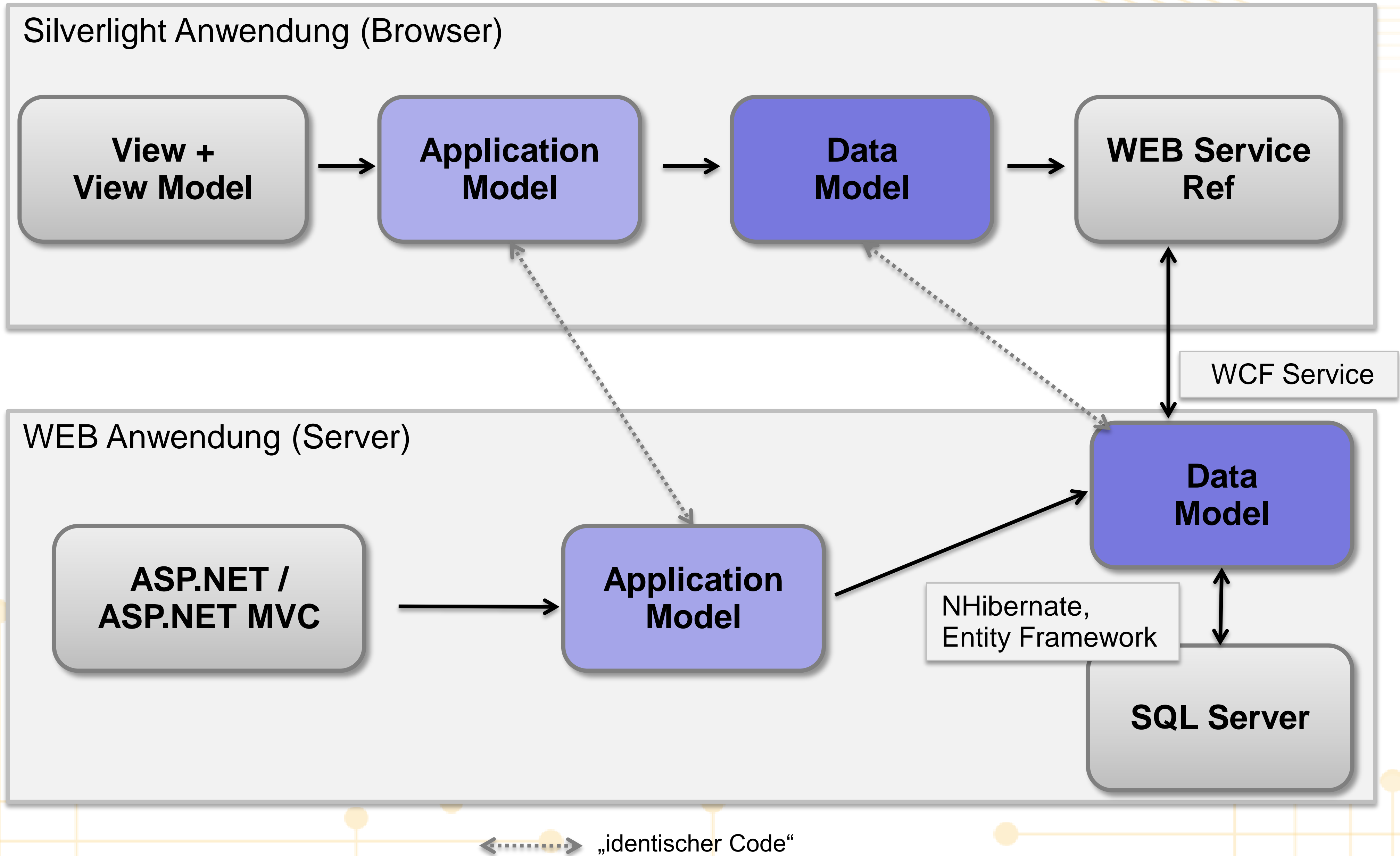
# Application Model (II)

- Muss ohne UI testbar sein
- Ggf. skriptbar (Ruby, Python, VB, C# 4, …)
- Application Model Domain spezifisch modellieren

# Model Trennung sinnvoll

- Datenbank oft vorgegeben (historisch bedingt)
- Abstraktion von konkreter Datenquelle
- Eigenständig testbar
- „Platz zum wachsen schaffen“ durch Trennung der Zuständigkeiten
- Entkopplung hilft auch bei Teamarbeit

# Vorteile der Model-Aufsplittung



# BEISPIEL

- WEB-basierte Informationssysteme
  - Erfassung mit Silverlight-Anwendung
  - Recherche mit ASP.NET-Anwendung

# Desktopanwendung

WPF Anwendung (Desktop)

...

Application Model

Data Model

SQL Server / Access

Alternative Quellen

WEB-Service Ref

WCF Service

WEB Service Anwendung

„gleicher Code“

BASTA! .NET, VISUAL STUDIO & MORE

# Tipp: Code Sharing

- Im Silverlight Projekt die Dateien aus .net Projekt referenzieren
- Bedingte Compilierung verwenden
- Partial Classes verwenden

# BEISPIEL

- Code Shareing
  - Application Model
  - Data Model

# Vorteile dieser Modellierung

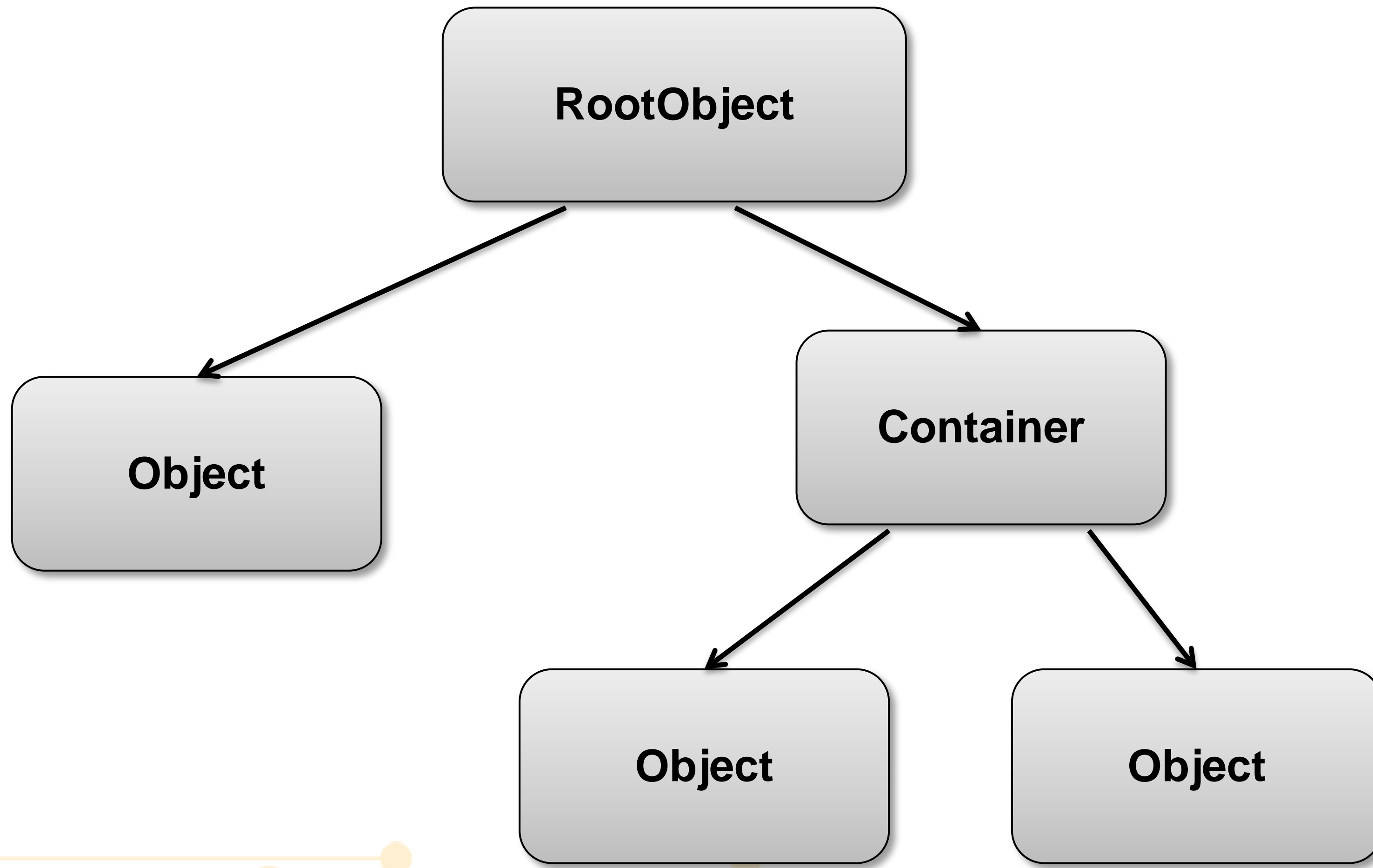
- Application Model läuft eigenständig
  - ggf. skriptbar
  - hervorragend testbar
- Datenzugriff austauschbar
- Ohne Silverlight testbar
- Ideal auch für Out-of-Browser Apps
  - Zwischenspeicherung im Isolated Storage

# BEISPIEL

- OPS-Explorer
  - Pufferung im Isolated Storage

# Tipp: Application Model

Baumstruktur (vergleichbar Word oder Excel)




BASTA! .NET, VISUAL STUDIO & MORE

# BEISPIEL

LOBS („Line Of Business Sample")

- ApplicationProcess (Root Object)

BASTA! .NET, VISUAL STUDIO & MORE

# Frameworks

- Application Model
  - Composite Application Guidance (CAG)
  - RIA Services

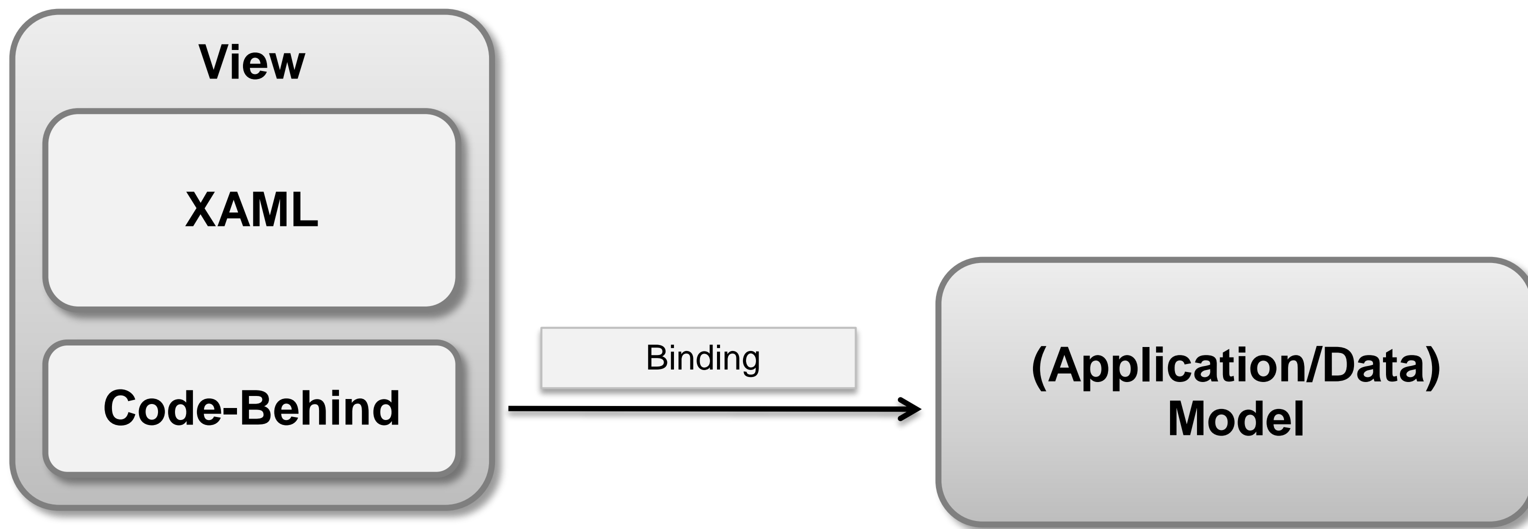
User Interface

# User Interface: MVVM Pattern

- Model – View – ViewModel
  - auch Model – View – Presentation Model
- Ersetzt MVC/MVP bei WPF / Silverlight
- „Optimiertes MVP für Data Binding“

# Beliebtes Design

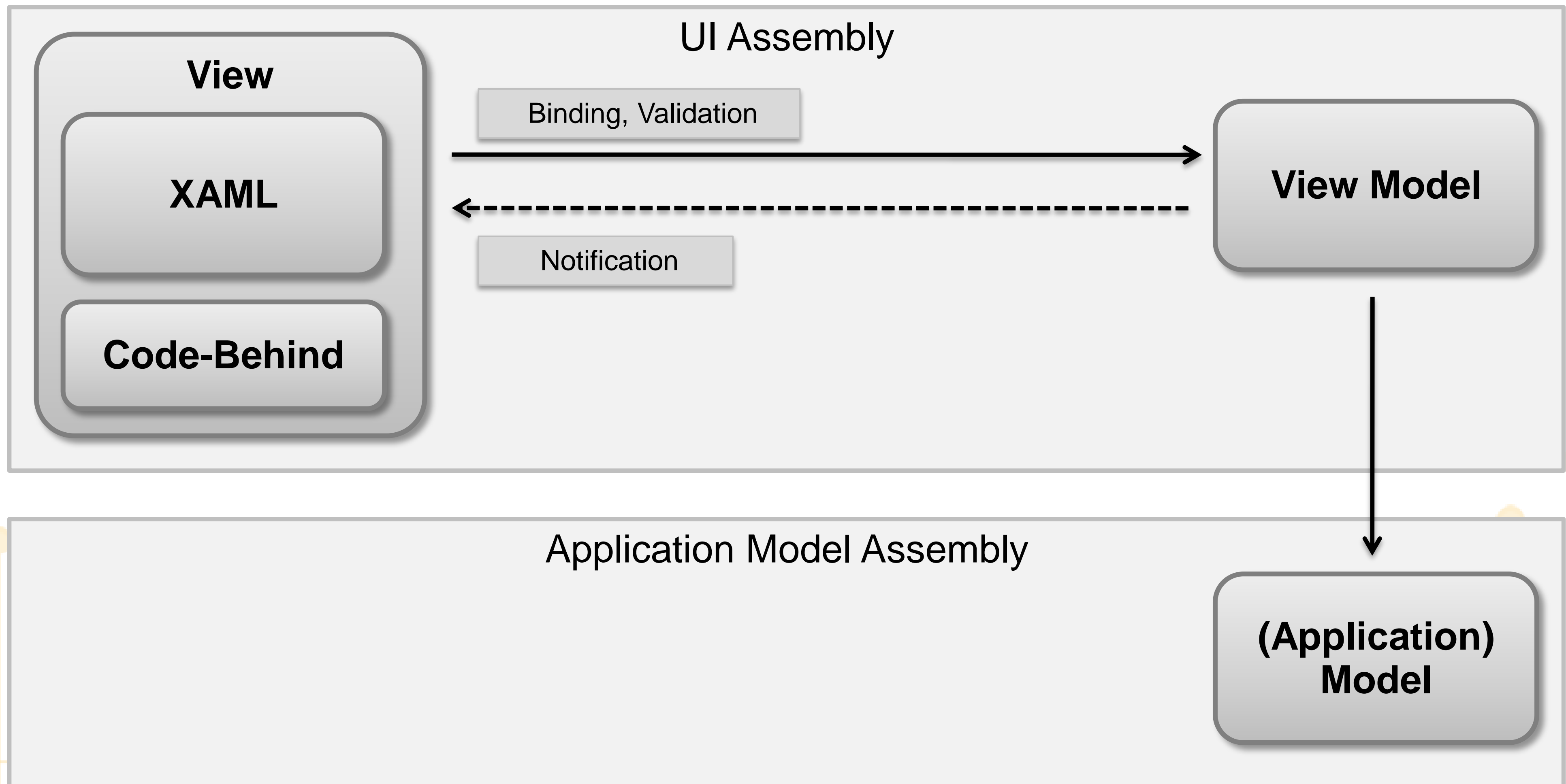
Direktes Binden ans Model




funktioniert, ist jedoch suboptimal

# Bessere Aufteilung

## Model – View – ViewModel



# Aufgaben des View Models

- Objekte für Data Binding bereitstellen
- Events/Commands
- Validierung
- Notifying des Views

- Verbindet fachliche Logik mit UI

# Aufgaben Code-Behind-Datei

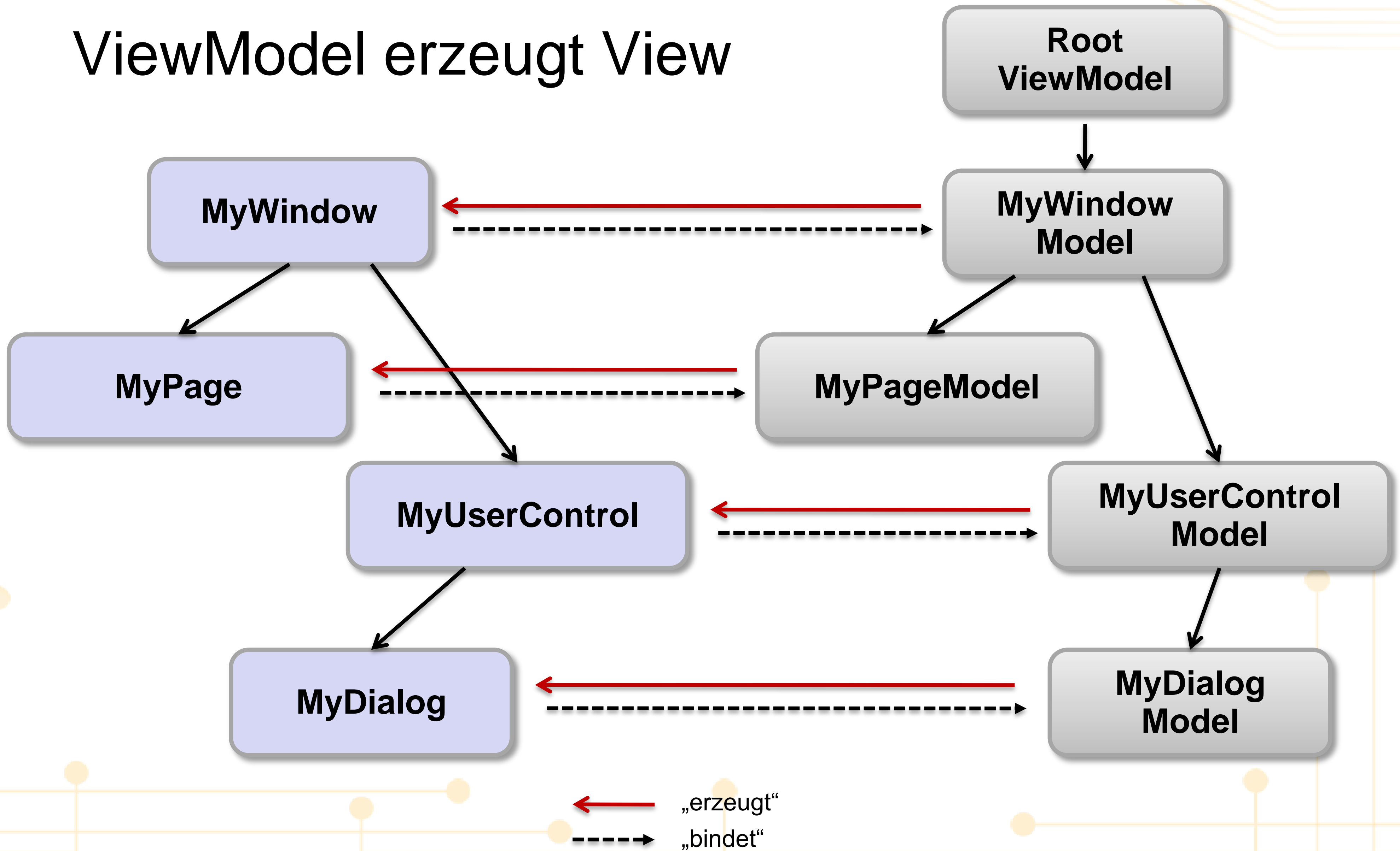
- Animationen
- Timer
- Styles / Templates / Theming
- Trigger / Behaviors
- Code für Custom Controls
- Silverlight: Event Handler

- „UI spezifischer, nicht fachlicher Code“

# Vorteil der Trennung

- ViewModel Klassen definieren Workflow
- ViewModel unabhängig von View Ausprägung (UserControl, Page, Dialog,…)
- WPF und Silverlight Code Sharing
  - ViewModel identisch
  - XAML + Code Behind unterschiedlich

# Tipp: ViewModel first!




BASTA! .NET, VISUAL STUDIO & MORE

# ViewModel Datenklasse (C#)

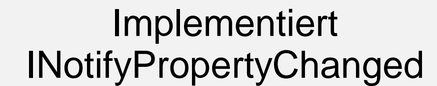
```
public class Employee : ViewModelBase
{
  public string LastName
  {
    get { return _lastName; }
    set { if (_lastName != value) { _lastName = value; RaisePropertyChanged(() => LastName); } }
  }
  …
```

Implementiert INotifyPropertyChanged

Tipp: Lambdas verwenden

# ViewModel Datenklasse (VB)

```
Public Class Employee
  Inherits ViewModelBase

Public Property LastName() As String
    Get
      Return Me._LastName
    End Get
    Set(ByVal value As String)
      If (Me._LastName <> value) Then
        Me._LastName = value
        MyBase.RaisePropertyChanged(Function() LastName)
    End If
    End Set
  End Property
  Private _LastName As String
```



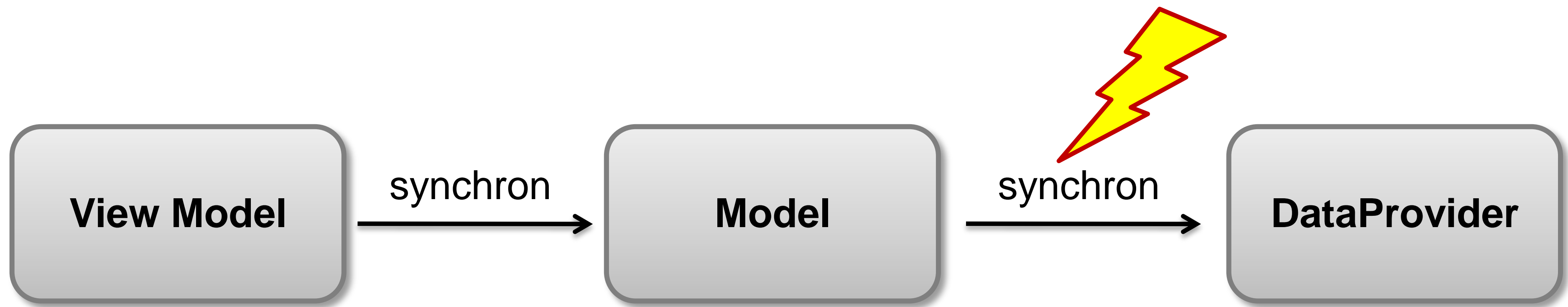
BASTA! .NET, VISUAL STUDIO & MORE

# BEISPIEL

ViewModel im LOBS

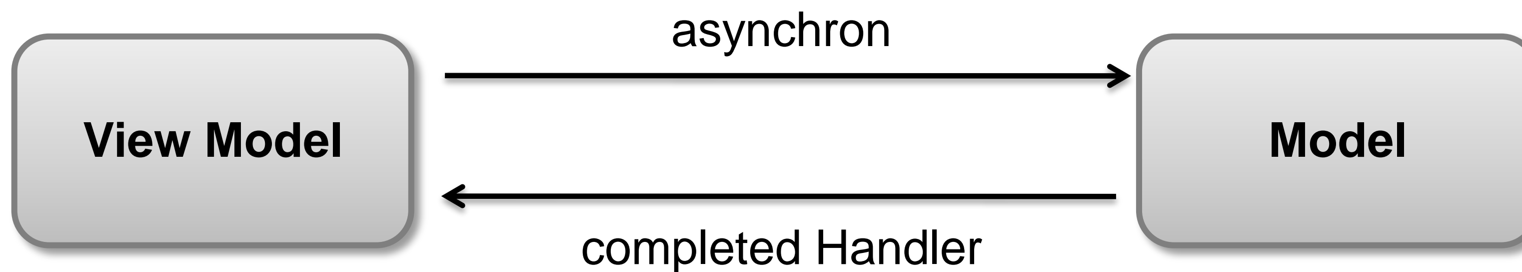
BASTA!
.NET, VISUAL STUDIO & MORE

# Synchrone Aufrufe




- In WPF üblich und kein Problem
- In Silverlight nicht möglich

# Asynchrone Aufrufe




- Diverse asynchrone Aufruftechniken verfügbar
  - Background Worker
  - Operations (WCF RIA Services)
  - Messaging
- Code wird dadurch komplizierter

# Synchrones Laden (C#)

## übersichtlich:

```
internal void LoadEmployeesSynchron()
{
  // Load employees synchronously
  var employees = ApplicationProcess.LoadEmployees();

  foreach (var employee in employees)
  {
    // Load photo of each employee
    employee.LoadPhoto();

    // Convert employee to view model
    _employees.Add(Employee.FromApplicationModel(employee));
  }
}
```

# Synchrones Laden (VB)

## übersichtlich:

```
Sub LoadEmployeesSynchron()
  ' Load employees synchronously
  Dim employees As List(Of ApplicationModel.Employee) = MyBase.ApplicationProcess.LoadEmployees
  Dim employee1 As ApplicationModel.Employee
  For Each employee1 In employees
    ' Load photo of each employee
    employee1.LoadPhoto()
    ' Convert employee to view model
    Me._employees.Add(Employee.FromApplicationModel(employee1))
  Next
End Sub
```

# Asynchrones Laden (C#)

## noch einigermaßen übersichtlich:

```
internal void LoadEmployeesAsynchron()
{
  var service =
    new AsyncServiceFunction<List<ApplicationModel.Employee>>(ApplicationProcess.LoadEmployees);
  service.ServiceCompleted += (sender, e) =>
  {
    var list = new List<Employee>();
    foreach (var item in e.Result)
    {
      Employee employee = Employee.FromApplicationModel(item);
      _employees.Add(employee);
      list.Add(employee);
    }
    // Load photos in a background thread
    ThreadPool.QueueUserWorkItem(obj => LoadEmployeePhotos(list), null);
  };
  service.InvokeServiceAsync();
}

void LoadEmployeePhotos(object obj)
{
   …
}
```

# Asynchrones Laden (VB)

## wird recht unübersichtlich:

```
Sub LoadEmployeesAsynchron()
    Dim service As New AsyncServiceFunction(Of List(Of ApplicationModel.Employee))
      (New Func(Of List(Of ApplicationModel.Employee))(AddressOf MyBase.ApplicationProcess.LoadEmployees))
      AddHandler service.ServiceCompleted, AddressOf LoadEmployeesCompleted
      service.InvokeServiceAsync()
    End Sub

Function LoadEmployeesCompleted(ByVal sender As Object, ByVal e As
      AsyncServiceCompletedEventArgs(Of List(Of ApplicationModel.Employee)))
  Dim list As New List(Of Employee)
  Dim item As ApplicationModel.Employee
  For Each item In e.Result
    Dim employee As Employee = employee.FromApplicationModel(item)
    Me._employees.Add(employee)
    list.Add(employee)
  Next
  ThreadPool.QueueUserWorkItem(Function(obj As Object) Me.LoadEmployeePhotos(list), Nothing)
  Return True
End Function
```

etc…

# Tipp: Asynchron vermeiden

Asynchrone Aufrufe zentralisieren

- Application Model am Anfang laden (async)
- Mit Application Model arbeiten (synchron)
- Am Ende speichern (async)

So könnte ein „Word für Silverlight“ arbeiten

# BEISPIEL

- Synchrones Laden
- Asynchrones Laden

# Fazit

- Schichtentrennung ist wichtig
- Anwendung so modellieren, dass sie wachsen kann
- Flexibel bleiben
  (Desktop, Browser, Out-of-Browser)
- Keine Angst vor zu vielen Assemblies

# Links

- Stefan.Lange@empira.de
- Unterlagen zu dieser Session
  www.st-lange.net